Introducción a la Programación

Guía Práctica 10
Archivos, Pilas, Colas y Diccionarios



Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

1. Archivos

Ejercicio 1. Implementar en python:

1. una función `contarlineas`(`in nombre_archivo : str`) → `int` que cuenta la cantidad de líneas de texto del archivo dado
2. una función `existePalabra`(`in palabra : str`, `in nombre_archivo : str`) → `bool` que dice si una palabra existe en un archivo de texto o no
3. una función `cantidadApariciones`(`in nombre_archivo : str`, `in palabra : str`) → `int` que devuelve la cantidad de apariciones de una palabra en un archivo de texto

Ejercicio 2. Dado un archivo de texto con comentarios, implementar una función `clonarSinComentarios`(`innombre_archivo : str`) que toma un archivo de entrada y genera un nuevo archivo que tiene el contenido original sin las líneas comentadas. Para este ejercicio vamos a considerar comentarios como aquellas líneas que tienen un caracter # como primer caracter de la línea, o si no es el primer caracter, se cumple que todos los anteriores son espacios.

Ejemplo:

```
# esto es un comentario
    # esto tambien
esto no es un comentario # esto tampoco
```

Ejercicio 3. Dado un archivo de texto, implementar una función que escribe un archivo nuevo (‘reverso.txt‘) que tiene las mismas líneas que el original, pero en el orden inverso.

Ejemplo: si el archivo original es

```
Esta es la primer linea.
Y esta es la segunda.
```

debe generar:

```
Y esta es la segunda.
Esta es la primer linea.
```

Ejercicio 4. Dado un archivo de texto y una frase (es decir, texto que puede estar separado por '\n'), implementar una función que la agregue al final del archivo original (sin hacer una copia).

Ejercicio 5. idem, pero agregando la frase al comienzo del archivo original (de nuevo, sin hacer una copia del archivo).

Ejercicio 6. Implementar una función que lea un archivo en modo *binario* y devuelva la lista de 'palabras legibles'. Vamos a definir una palabra legible como

- secuencias de texto formadas por numeros, letras mayusculas/minusculas y los caracteres ' '(espacio) y '_'(guion bajo)
- que tienen longitud >= 5

Una vez implementada la función, probarla con diferentes archivos binarios (.exe, .zip, .wav, .mp3, etc).
Referencia: https://docs.python.org/es/3/library/functions.html#open

Ejercicio 7. Implementar una función que lea un archivo de texto separado por comas (comma-separated values, o .csv) que contiene las notas de toda la carrera de un grupo de alumnos y calcule el promedio final de un alumno dado. La función `promedioEstudiante`(in `lu` : `str`) → `float`. El archivo tiene el siguiente formato:

```
nro de LU (str), materia (str), fecha (str), nota (float)
```

2. Pilas

Ejercicio 8. Implementar una función `generarNrosAlAzar`(in `n` : `int`, in `desde` : `int`, in `hasta` : `int`) → `list`[`int`] que genere una lista de n numeros enteros al azar en el rango $[desde, hasta]$. Pueden user la función `random.sample()`

Ejercicio 9. Usando la función del punto anterior, implementar otra función que arme una pila con los numeros generados al azar. Pueden usar la clase `LifoQueue`() que es un ejemplo de una implementación básica:

```
from queue import LifoQueue as Pila

p = Pila()
p.put(1) # apilar
elemento = p.get() # desapilar
p.empty() # vacia?
```

Ejercicio 10. Implementar una función `cantidadElementos`(in `p` : `pila`) → `int` que, dada una pila, cuente la cantidad de elementos que contiene.

Ejercicio 11. Dada una pila de enteros, implementar una función `buscarElMaximo`(in `p` : `pila`) → `int` que devuelva el máximo elemento.

Ejercicio 12. Implementar una función `estaBienBalanceada`(in `s` : `str`) → `bool` que dado un string con una formula aritmética sobre los enteros, diga si los paréntesis estan bien balanceados. Las fórmulas pueden formarse con:

- los numeros enteros
- las operaciones basicas +, −, x y /
- parentesis
- espacios

Entonces las siguientes son formulas aritméticas con sus paréntesis bien balanceados:

```
1 + (2 x 3 − (20 / 5))
10 ∗ (1 + (2 ∗ (−1)))
```

Y la siguiente es una formula que no tiene los paréntesis bien balanceados:

```
1 + ) 2 x 3 ( ()
```

3. Colas

Ejercicio 13. Usando la función `generarNrosAlAzar`() definida en la sección anterior, implementar una función que arme una cola de enteros con los numeros generados al azar. Pueden usar la clase `Queue`() que es un ejemplo de una implementación básica:

```
from queue import Queue as Cola

c = Cola()
c.put(1) # encolar
elemento = c.get() # desencolar()
c.empty() # vacia?
```

Ejercicio 14. Implementar una función `cantidadElementos`(in c : cola) → int que, dada una cola, cuente la cantidad de elementos que contiene. Comparar con la versión usando pila.

Ejercicio 15. Dada una cola de enteros, implementar una función `buscarElMaximo`(in c : cola) → int que devuelva el máximo elemento. Comparar con la versiń usando pila.

Ejercicio 16. Bingo: un cartón de bingo contiene 12 números al azar en el rango $[0, 99]$.

1. implementar una función `armarSecuenciaDeBingo`() → Cola[int] que genere una cola con los números del 0 al 99 ordenados al azar.
2. implementar una función `jugarCartonDeBingo`(in carton : list[int], in bolillero : cola[int]) → int que toma un cartón de Bingo y una cola de enteros (que corresponden a las bolillas numeradas) y determina cual es la cantidad de jugadas de ese bolillero que se necesitan para ganar.

Ejercicio 17. Vamos a modelar una guardia de un hospital usando una cola donde se van almacenando los pedidos de atención para los pacientes que van llegando. A cada paciente se le asigna una prioridad del 1 al 10 (donde la prioridad 1 es la mas urgente y requiere atención inmediata) junto con su nombre y la especialidad medica que le corresponde.

Implementar la función `nPacientesUrgentes`(in c : Cola[(int, str, str)]) → int que devuelve la cantidad de pacientes de la cola que tienen prioridad en el rango $[1, 3]$.

4. Diccionarios

Para esta sección vamos a usar el tipo `dict` que nos provee python:

Ejercicio 18. Leer un archivo de texto y agrupar la cantidad de palabras de acuerdo a su longitud. Implementar la función `agruparPorLongitud`(in nombre_archivo : str) → dict que devuelve un diccionario {longitud_en_letras : cantidad_de_palabras}.

Ej el diccionario

```
{
        1: 2,
        2: 10,
        5: 4
}
```

indica que se encontraron 2 palabras de longitud 1, 10 palabras de longitud 2 y 5 palabras de longitud 4. Para este ejercicio vamos a considerar palabras a todas aquellas secuencias de caracteres que no tengan espacios en blanco.

Ejercicio 19. Volver a implementar la función que calcula el promedio de las notas de los alumnos, pero ahora devolver un diccionario {libreta_universitaria : promedio} con los promedios de todos los alumnos.

Ejercicio 20. Implementar la función `laPalabraMasFrecuente`(in nombre_archivo : str) → str que devuelve la palabra que más veces aparece en un archivo de texto.